A METHOD FOR DETECTING AND DETERMINING INTERSECTIONS OF GEOMETRIC OBJECTS

BY

PAUL JEFFREY LEWICKI

A DISSERTATION PRESENTED TO THE GRADUATE SCHOOL
OF THE UNIVERSITY OF FLORIDA IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

UNIVERSITY OF FLORIDA

1991

TABLE OF CONTENTS

ABSTRACT iv

CHAPTERS

1 PROBLEM STATEMENT 1

2 CURRENT APPROACHES TO THE PROBLEM 3

3 SCOPE OF THIS WORK 8

4 OBJECT REPRESENTATIONS 10

Background Theory 10
Mathematical Representation of an Object 13

5 OBJECT INTERSECTIONS 16

Requirements for Intersection 16
The Form of the Solution 17

6 DETECTING INTERSECTIONS 18

A Simple Case 20
Underconstrained Systems 23
Checking for Intersection 27
Embedded Problems and Overconstrained Problems 30

7 DETERMINING INTERSECTIONS 32

Another Look at the New System 32
Solvable Equation Sub-Sets 32
Elimination of Non-Extreme Points 36
Object Facial Structure List Construction 37
Transformation to Coordinate Space 39

8 IMPLEMENTATION 40

Programming Environment 40
The Programs 40
Computational Cost 42

9 CONCLUDING REMARKS 47

Summary 47
Suggestions for Further Work 47

REFERENCES 49

BIOGRAPHICAL SKETCH 50

Abstract of Dissertation Presented to the Graduate School
of the University of Florida in Partial Fulfillment of the
Requirements for the Degree of Doctor of Philosophy

A METHOD FOR DETECTING AND DETERMINING INTERSECTIONS OF GEOMETRIC OBJECTS

BY

PAUL JEFFREY LEWICKI

December, 1991

Chairman: Gary K. Matthew
Major Department: Mechanical Engineering

A method for detecting the intersection of convex objects in two- and three-dimensional space which have a finite number of extreme points is presented. The method is based on a definition of an object that is drawn from the theory of convex sets. Using this definition, the conditions of intersection are expressed as a set of linear equations and a set of inequalities. From this set of equations a new set of equations is produced which allows for the existence of an intersection to be detected. The method is extended to determine the object of intersection. The result of the determination is a list of the extreme points and lists describing the facial structure of the object of intersection.

CHAPTER 1
PROBLEM STATEMENT

This work will deal with geometric objects and their intersections. The term geometric object refers to a set of points in some space. Examples of geometric objects are the points on a line segment, the points enclosed by a triangle, or the points enclosed by the sides of a tetrahedron. Each of these geometric objects is a closed region of space. While the number of points within the object can be infinite, the objects themselves are of finite extent. For example, a line has infinite extent, but a line segment, while consisting of an infinite number of points, is of limited extent. Geometric objects have a boundary and an interior. All of the points that lie in the interior and all of the points that lie on the boundary are considered to be elements of the object. So that, when we refer to a square in a plane, we mean the points enclosed by the lines that connect the four vertices, as well as the lines themselves (and, indeed, the vertices themselves). Geometric objects can be classified as convex or non-convex. If every element of a line drawn between any two points in the object is a member of the object, then it is convex. If such a line has any points which are not members of the object then the object is non-convex.

The intersection of two objects is defined as the set of points which are members of both of the objects. In some situations the boundaries of two objects will have common points. But in other situations, one object may lie completely

inside the other. This will be referred to as inclusion. Both the situation where the boundaries intersect and when we have inclusion are considered to be intersections.

In certain applications, one of the fundamental operations is to determine if two geometric objects intersect one another. This is the detection problem. Some such applications are determining the visibility of objects in computer graphics, determining collisions in robot path planning, and checking clearances in mechanical assemblies. This can be stated as a predicate "does object A intersect object B?" The result of this operation is a true or false value. The focus of this research is whether by treating geometric objects as entities in themselves, this predicate can be solved directly using algebraic means.

Another problem is to determine the actual object of intersection. Applications which would require the solution of this problem are constructive solid geometry and tool path planning. The result of this operation is a definition of the object of intersection.

CHAPTER 2
CURRENT APPROACHES TO THE PROBLEM

There have been many investigations of this problem, taking several different approaches. These approaches have used various definitions of an object, and various methods of detecting and determining intersections. Some of the methods are applicable only to two-dimensional problems, while others apply to three-dimensional problems as well. Some investigators look only at stationary objects while others also look at moving objects.

Boyse [1] looks at the problem from the application of mechanical assembly and manufacturing. His approach defines solids in terms of boundary surfaces or faces, which are in turn defined by their perimeters, which are in turn defined by a set of edges, which are in turn defined by a pair of vertices. This hierarchy builds up solid objects from simpler objects. Testing for intersections is done by testing all face/edge pairs of the two objects, and further testing for inclusion of one object inside another. He also looks for collisions between a stationary object and a moving object. The moving object sweeps out a space curve or a surface which he then checks for intersection with the faces or edges of the stationary object.

Chazelle and Dobkin [2] present algorithms for detecting and finding intersections in two and three dimensions. For the two-dimensional case, polygons are represented as arrays with the vertices given in clockwise order, lines

by a pair of points. They start with an algorithm for the intersection of a line with a convex polygon (Algorithm IGL). Consider two points, x and v, and a line in a plane. They define a function h(x,L,v) as the orthogonal distance from the point x to the line L, with respect to the point v, that is h(x,L,v) is positive if x and v are on the same side of L and negative if they are on opposite sides of L. They use a Fibonacci search to find the vertex of the polygon that has the minimum h(x,L,p_1), where p_1 is the first vertex in the array. If this distance is zero or negative then the line intersects the polygon. They state that the extension to the case in which L is a line segment "does not increase the time bound" [2, p. 4] of the solution, but they do not give details of the solution. They then present an algorithm for detecting the intersection of two convex polygons. They state: "If we could determine the side of P closest to Q (and vice versa), we would be able to reduce the problem to a small number of tests of segment intersections. Our method reduces the number of remaining edges of one of the polygons by a factor of 2 at each iteration" [2, p. 4]. The algorithm (IGG) they present contains several inclusion tests, both of a point within a polygon and of a point within a line segment. The algorithm also contains a Fibonacci search of a set of oriented angles. This reduces each polygon to a contiguous polygonal line. Then a recursive function is called which for each call reduces the size of the polygonal lines until either an intersection is found or excluded. They use a special representation for three dimensional objects. Each polyhedron is represented as a set of parallel convex polygons--determined during a preprocessing step. These polygons, called preprocessing polygons, consist of a cross-section of the polyhedron for each vertex. Each cross-section is the

intersection of the polyhedron with a plane parallel to the xy-plane passing through the vertex. This reduces a polyhedron P of p vertices to a set of p (or fewer) convex polygons $P_1, ..., P_p$ and p-1 convex drums. A convex drum is a polyhedron with all of its vertices lying on two parallel faces. Using this representation they then present algorithms for the intersection of a polyhedron and a plane, the intersection of a polyhedron and a polygon, and the intersection of two polyhedra.

Shamos and Hoey [3] present algorithms for determining whether any two line segments out of a number of line segments in a plane intersect. Their algorithm involves a plane-sweeping method, that is, the plane is swept by a line and the order in which the endpoints of the line segments are encountered is used to determine whether they intersect.

Nievergelt and Preparata [4] also present a set of plane-sweep algorithms for finding intersections of line segments and polygons.

Muller and Preparata [5] present an algorithm that given two convex polyhedra (i) tests whether their intersection is empty, and (ii) if so finds a separating plane, while (iii) if not finds a point in the intersection and explicitly constructs their intersection polygon. Their method of finding the intersection of the two polyhedra is to find the projections of each of them on a plane, which will form two polygons, and then to use Shamos and Hoey's algorithm to determine if the two polygons intersect. If the two do not intersect then the two polyhedrons do not intersect. If the two polygons do intersect, they then determine the line segments in the polyhedrons that project into the intersection of the two

polygons. If these two line segments intersect then they have determined an intersection point, but if not they project the near-sides of the two polyhedrons onto the xy-plane to obtain two planar straight-line-graphs. They then use a searching technique to examine the planar graph to find faces of the polyhedron that intersect.

Dobkin and Kirkpatrick [6] extend the previous work of Chazelle and Dobkin. In their methods convex polygons and polyhedra are defined through a hierarchy of descriptions each refining previous definitions. They start with coarse descriptions of objects and as they progress through the steps of their algorithm finer descriptions of smaller portions of the objects are given. The algorithm focuses on those portions of the objects which are relevant to possible intersection.

Ganter and Uicker [7] present a method of detecting the collision of moving objects. Objects are represented using solid modeling techniques. Static intersections are detected by using constructive solid geometry techniques to find the volume of intersection. Dynamic intersections are detected by forming a new object which is the volume that one of the objects would sweep out as it moves relative to a reference object. The collision is detected if there is a static intersection between the new object and the reference object.

Lozano-Perez [8] presents a method of planning a path without collisions using a configuration space method. In the configuration space method a moving object with a fixed orientation is reduced to a point while at the same time the obstacles are expanded in such a manner that if the point does not intersect the

expanded obstacles then the original object would not either. In this way the intersection of two objects is reduced to determining if a single point is inside of an obstacle.

CHAPTER 3
SCOPE OF THIS WORK

This work will not attempt to solve all types of intersections, but rather will address only certain types of objects and certain types of intersection between them. The objects to be considered are one, two, and three dimensional. The one dimensional case is academic, while the two and three dimensional cases are for practical consideration. A single method which can handle all three dimensions will be developed, which will require a method of representing objects that is uniform across all dimensions. Only convex objects with a finite number of vertices will be considered. This class of objects includes the polygons in two dimensions and the polyhedrons in three dimensions. Not included are objects such as a circular disk or a segment of sphere which are convex but do not have a finite number of vertices. The number of vertices, however, is not limited except by practical considerations such as storage and computation time. Therefore the method will work for an approximation to a circular disk, but the better the approximation, the greater the cost. The method will only consider static objects. No attempt will be made to consider objects that change relative positions over time.

A single method that is applicable for one, two, and three dimensions has been developed. The method detects the intersection between two objects of the same dimension. If the intersection of more than two objects is desired, then the

method can be applied to two of the given objects and the method then applied to the resulting object of intersection and another of the given objects. In this way the intersection of a group can be determined. An extension of the method allows the determination of the object of intersection.

CHAPTER 4
OBJECT REPRESENTATIONS

Solid objects are sets of points, that is they are subsets of the space in which they exist. Convex objects are convex sets of points. In order to work with convex objects and perform the operations of detecting and determining the intersections between them, it is desired to have a mathematical definition of a convex object. The definition that I have chosen to use is drawn from the theory of convex sets.

Background Theory

This section will summarize the theory of convex sets as presented in Brondsted [9].

Affine Combinations

A subset A of R^d is an affine subspace if, and only if, the following holds: $\lambda_1 x_1 + \lambda_2 x_2$ is in A for all $x_1, x_2 \in A$ and all $\lambda_1, \lambda_2 \in R$ with $\lambda_1 + \lambda_2 = 1$. An affine combination of points $x_1, \ldots, x_n$ from R^d is a linear combination $\lambda_1 x_1 + \cdots + \lambda_n x_n$, where $\lambda_1 + \cdots + \lambda_n = 1$. We shall write

$$\sum_{i=1}^{n}{}^{a} \lambda_i x_i$$

to indicate that the linear combination $\lambda_1 x_1 + \cdots + \lambda_n x_n$ is in fact an affine combination. The intersection of any family of affine subspaces of R^d is again an

affine subspace of R^d. For any subset M of R^d, the affine hull *aff* M is the set of all affine combinations of points from M. An n-family $(x_1, \ldots, x_n)$ of points from R^d is said to be affinely independent if a linear combination $\lambda_1 x_1 + \cdots + \lambda_n x_n$ with $\lambda_1 + \cdots + \lambda_n = 0$ can only have the value 0 when $\lambda_1 = \cdots = \lambda_n = 0$. An affine basis of an affine space A is an affinely independent n-family $(x_1, \ldots, x_n)$ of points from A such that $A = aff\{x_1, \ldots, x_n\}$. The dimension *dim* A of a non-empty affine space A is the dimension of the linear subspace L such that $A = x + L$. An affinely independent n-family of points from A is an affine basis of A if and only if $n = dim\ A + 1$. For any subset M of R^d, there exists an affinely independent family $(x_1, \ldots, x_n)$ of points from M such that *aff* M is the set of all affine combinations

$$\sum_{i=1}^{n} \lambda_i x_i$$

of $x_1, \ldots, x_n$. This statement shows that to generate *aff* M it suffices to take all affine combinations of the fixed points $x_1, \ldots, x_n$ from M. Further, each point in *aff* M has a unique representation as an affine combination of $x_1, \ldots, x_n$.

Convex Combinations

A subset C of R^d is called a convex set if $\lambda_1 x_1 + \lambda_2 x_2$ belongs to C for all $x_1, x_2 \in R$ with $\lambda_1 + \lambda_2 = 1$ and $\lambda_1, \lambda_2 \geq 0$. When x_1 and x_2 are distinct points from R^d, then the set

$$[x_1, x_2] := \{\lambda_1 x_1 + \lambda_2 x_2 \mid \lambda_1, \lambda_2 \geq 0, \lambda_1 + \lambda_2 = 1\}$$

$$= \{(1-\lambda)x_1 + \lambda x_2 \mid \lambda \in [0,1]\}$$

is called the closed segment between x_1 and x_2. With this notation, a set C is convex if, and only if, the closed segment between any two points of C is contained in C. By a convex combination of points $x_1, \ldots, x_n$ from R^d we mean a linear combination $\lambda_1 x_1 + \cdots + \lambda_n x_n$, where $\lambda_1 + \cdots + \lambda_n = 1$ and $\lambda_1, \ldots, \lambda_n \geq 0$. Every convex combination is also an affine combination. We shall write

$$\sum_{i=1}^{n}{}^{c} \lambda_i x_1$$

to indicate that the linear combination $\lambda_1 x_1 + \cdots + \lambda_n x_n$ is in fact a convex combination. A subset C of R^d is convex if and only if any convex combination of points from C is again in C. The intersection of any family of convex sets in R^d is again convex. For any subset M of R^d, the convex hull *conv* M is the set of all convex combinations

$$\sum_{i=1}^{n}{}^{c} \lambda_i x_1$$

such that $(x_1, \ldots, x_n)$ is an affinely independent family of points from M. In other words, in order to generate *conv* M we need not take all convex combinations of points from M, it suffices to take those formed by the affinely independent families of points from M. On the other hand, no fixed family of points from M will suffice. For any subset M of R^d with $dim(aff\ M) = n$, the convex hull *conv* M is the set of all convex combinations of precisely $n + 1$ points from M. By a convex polytope, or simply a polytope, we mean a set which is the

convex hull of a non-empty finite set $\{x_1, \ldots, x_m\}$. A polytope S with the property that there exists an affinely independent family $(x_1, \ldots, x_n)$ such that $S = conv\{x_1, \ldots, x_n\}$ is called a simplex and the points $x_1, \ldots, x_n$ are called the vertices of S. If $x_1, \ldots, x_n$ are the vertices of a simplex S, then by the affine independence each point in $aff\{x_1, \ldots, x_n\}$ has a unique representation as an affine combination of $x_1, \ldots, x_n$, in particular, each point in $conv\{x_1, \ldots, x_n\}$ has a unique representation as a convex combination of $x_1, \ldots, x_n$. Simplices are the only polytopes having a "convex basis" as described above.

Facial Structure of a Convex Set

Let C be a closed convex set in R^d. A convex subset F of C is called a face of C if for any two distinct points $y, z \in C$ such that $]y,z[\cap F$ is non-empty, we actually have $[y,z] \subset F$. A point $x \in C$ is called an extreme point of C if $\{x\}$ is a face. A face F of C is called a k-face if $dim\ F = k$. Thus, the 0-faces are the extreme points. A facet of C is a face F with $0 \leq dim\ F = dim\ C - 1$. The intersection of any set of faces of C is again a face of C.

For example, consider a three dimensional cube as shown in Figure 1. The 2-faces or facets of the cube are the sides, the 1-faces are the edges, and the 0-faces are the corners. The intersection of any two facets is a 1-face. The intersection of any two 1-faces is a 0-face.

Mathematical Representation of an Object

Using the definition of a convex set as given by Brondsted, an object is defined as the set of points that satisfy the following requirements:

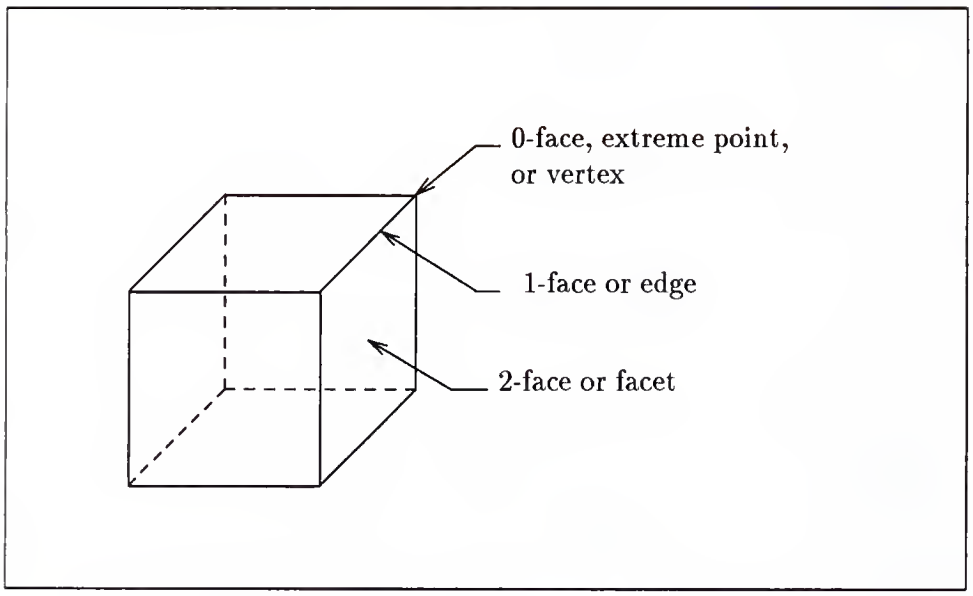

Figure 1. Facial Structure of a Cube

$$\lambda_1 P_1 + \lambda_2 P_2 + \cdots + \lambda_n P_n = X,$$
$$\lambda_1 + \lambda_2 + \cdots + \lambda_n = 1,$$
$$\lambda_1, \lambda_2, \cdots, \lambda_3 > 0.$$

This definition provides a relationship between points in the xyz-space where the object exists, which will be referred to as the coordinate space, and points in the λ-space, which will be referred to as the parameter space. The first equation expresses any point X as a linear combination of the set of points $\{P_i\}$ which are the extreme points or vertices of the object. This is therefore a vector equation and represents either one, two, or three scalar equations depending on the dimension of the coordinate space. By requiring that the values of the λ's also satisfy the second and third equations the point X must lie in the convex combination of the extreme points. Each set of n λ's (a point in the n-dimensional parameter space) which satisfy the second and third equation define a point in the coordinate space. Unless the number of extreme points in the

object is less than or equal to $d+1$, where d is the dimension of the coordinate space, a given point in coordinate space may have more than one set of λ's that satisfy all of the stated conditions.

This definition is sufficient to define a convex object but additional information is needed to efficiently solve for the object of intersection. This additional information has the following form:

1. Edges: a list of the pairs of points which form the edges of the object.
2. Faces: a list of the triples of points which tile the boundary of the object with non-overlapping triangles.
3. Composite Faces: a list of which of the faces belong to coplanar surfaces.
4. Volumes: a list of the quartets of points which divide the object into non-overlapping tetrahedrons.

These lists are not all needed for objects of all dimensions. The number of faces, composite faces, and volumes of a one-dimensional object are all zero. For a two-dimensional object the number of volumes is zero and the number of composite faces is one.

For this work it is assumed that we start with a pair of correctly formed objects. In a correctly formed object the list of points contains only the extreme points of the convex object, and the lists describing the facial structure of the object are all correct.

CHAPTER 5
OBJECT INTERSECTIONS

Requirements for Intersection

The intersection of two objects A and B is the set of points X, such that X is a member of both A and B. Therefore any point X in the intersection must simultaneously satisfy the defining relations of both objects. Setting the two vector equations for X equal to each other produces a new vector equation:

$$\lambda_1 A_1 + \cdots + \lambda_n A_n = \lambda_{n+1} B_1 + \cdots + \lambda_{n+m} B_m,$$

where A and B are the extreme points from object A and B respectively. In addition there are the two λ equations :

$$\lambda_1 + \lambda_2 + \cdots + \lambda_n = 1,$$

$$\lambda_{n+1} + \lambda_{n+2} + \cdots + \lambda_{n+m} = 1,$$

and the set of inequalities:

$$\lambda_1, \lambda_2, \cdots, \lambda_{n+m} \geq 0.$$

This is a set of equations and inequalities only in terms of the λ's. Working with the problem in parameter space rather than coordinate space simplifies the solution. The solution to this set of relations is the set of values of the λ's which satisfy all of the constraints. The relations can be divided into two groups: a system of linear equations, and the set of inequalities. For most intersection

problems the set of linear equations will be underconstrained, that is, there will be more unknowns than equations. This says that there is an infinite number of solutions to the system of equations. Of course, this is to be expected since the object of intersection will usually contain an infinite number of points. However, not all solutions to the system of equations will be a point in the intersection of the two objects. The solutions which form the intersection are those that also satisfy the additional constraints imposed by the set of inequalities.

The Form of the Solution

We are considering two separate tasks:

1. detecting if the intersection is non-empty, that is, detecting if the solution to the system of equations contains any points which also satisfy the set of inequalities,

2. determining the object of intersection.

The result of task 1 is a true or false indication of whether an object of intersection exists. The result of task 2 is a definition of the object of intersection, in the form of the given objects. This will require finding the limiting values of the λ's that define points in the solution set. These limiting values will correspond to the extreme points of the object of intersection in the xyz-space.

CHAPTER 6
DETECTING INTERSECTION

The first task to be dealt with is detecting whether an intersection exists. The method used depends on looking at the equations from the mathematical statement of the intersection as representations of geometrical objects. A simple case of using this interpretation in the solution of a system of equations is used to illustrate the concepts involved, leading to a criterion for intersection detection. Then the method is extended to the problem of underconstrained systems of equations such as are produced by the intersection of objects. The criterion for intersection is then developed for the underconstrained system. The chapter will end with a discussion of special cases, why they occur and how they can be handled.

The method of detecting intersections is based on a geometrical rather than algebraic interpretation of the equations and inequalities that are given in the statement of intersection. An equation can be thought of as a representation of a hyperplane. A hyperplane is an (n-1)-dimensional object in an n-dimensional space. For example, in a two-dimensional space the hyperplanes are lines, in a three-dimensional space the hyperplanes are planes. In a four-dimensional space the hyperplanes would be objects of three dimensions. In solving a system of equations we seek a set of points that satisfy all of the equations. But we can think of this in terms of the hyperplanes that the equations represent. Then the

solution is the set of points which is common to all of the hyperplanes; that is, their intersection. Now at this point it might seem that we are back where we started from — looking for the intersections of objects. But there is a difference. The objects that we started with were limited; for instance, a portion of a plane, while the hyperplanes we are now dealing with are unlimited.

So far we have dealt with the equations in our problem. How do we deal with the inequalities? The inequalities, together with the two λ equations which require that the sum of the parameter values for each object be equal to one, impose the constraint that the acceptable points of intersection must have parameter values that fall in the range between zero and one, inclusive. In terms of the geometric interpretation of the problem this is a requirement that the object of intersection lie inside of an n-dimensional unit cube. The three-

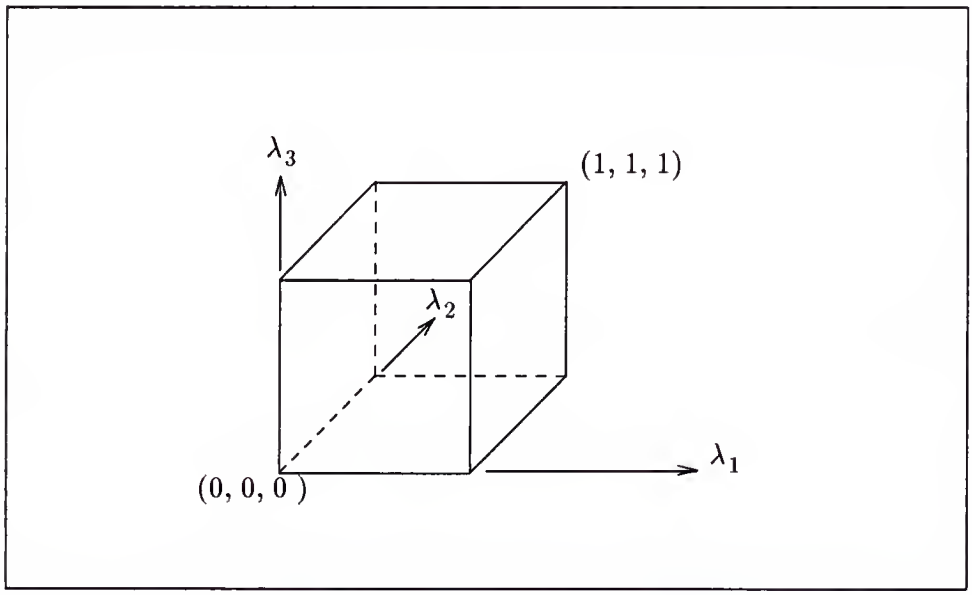

Figure 2. The Three-Dimensional Critical Region

dimensional critical region is shown in Figure 2. This n-dimensional unit cube

will be referred to as the critical region. If the intersection of the hyperplanes passes through the critical region then there are points common to both objects. If the intersection of the hyperplanes does not pass through the critical region the there are no points common to both objects.

A Simple Case

Looking at these concepts in terms of a simple set of constrained equations will illustrate how interpreting the equations as a representation of geometry leads to a criterion for detecting intersections. Consider a constrained system of equations, say, three equations in three unknowns.

$$\begin{aligned}
\alpha_{11}x_1 + \alpha_{12}x_2 + \alpha_{13}x_3 &= \beta_1 \\
\alpha_{21}x_1 + \alpha_{22}x_2 + \alpha_{23}x_3 &= \beta_2 \\
\alpha_{31}x_1 + \alpha_{32}x_2 + \alpha_{33}x_3 &= \beta_3
\end{aligned}$$

If we assume that the coefficient matrix $[\alpha]$ has rank three then we know that the system of equations has a single solution. If we approach the solution from the point of view of linear algebra then we think of transforming the matrix $[\alpha]$ into a new matrix $[\alpha']$, such that the new matrix is diagonal. The new system can be written as

$$\begin{aligned}
|\alpha_1\alpha_2\alpha_3|\,x_1 + 0x_2 + 0x_3 &= |\beta\alpha_2\alpha_3| \\
0x_1 + |\alpha_1\alpha_2\alpha_3|\,x_2 + 0x_3 &= |\alpha_1\beta\alpha_3| \\
0x_1 + 0x_2 + |\alpha_1\alpha_2\alpha_3|\,x_3 &= |\alpha_1\alpha_2\beta|
\end{aligned}$$

where each of the terms of the form $|\alpha_1\alpha_2\alpha_3|$ is defined as

$$|\alpha_1\alpha_2\alpha_3| = \begin{vmatrix} \alpha_{11} & \alpha_{12} & \alpha_{13} \\ \alpha_{21} & \alpha_{22} & \alpha_{23} \\ \alpha_{31} & \alpha_{32} & \alpha_{33} \end{vmatrix}.$$

The new system of equations leads directly to the solution set of the original system of equations.

If we now think in terms of hyperplanes rather than equations, how do we describe the process of solving the system of equations? Each of the three equations in the original system represents a hyperplane in the three-dimensional

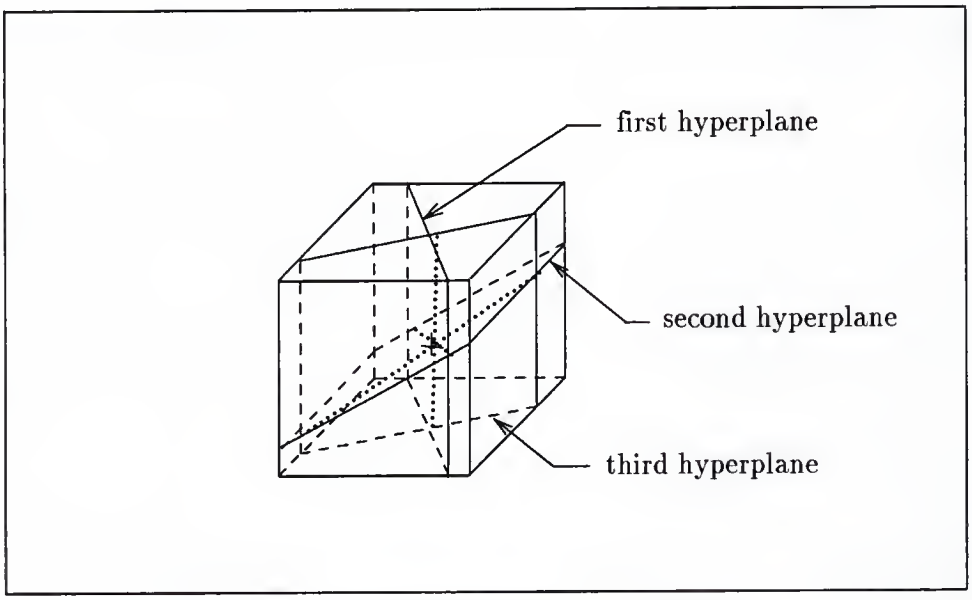

Figure 3. Three General Hyperplanes

space. A possible example is shown in Figure 3. Likewise, each of the three equations in the new system represents a hyperplane. The new hyperplanes are obtained by forming linear combinations of the original hyperplanes. As long as we replace a hyperplane with one obtained by taking a linear combination of hyperplanes in the set, then we will have a new set that has the same intersection.

Why is the new set of hyperplanes considered to be a solution whereas the original set is not? In both cases the intersection of the three hyperplanes is the solution set of the system of equations. But in the second case, the hyperplanes have a particular relationship with the coordinate axes and with each other. The hyperplane of the form

$$\alpha_{11}' x_1 + 0x_2 + 0x_3 = \beta_1'$$

is perpendicular to the x_1 axis and parallel to the x_2 and x_3 axes. Likewise, each of the other equations is perpendicular to one axis and parallel to the other two. This means that the three hyperplanes are also perpendicular to each other as

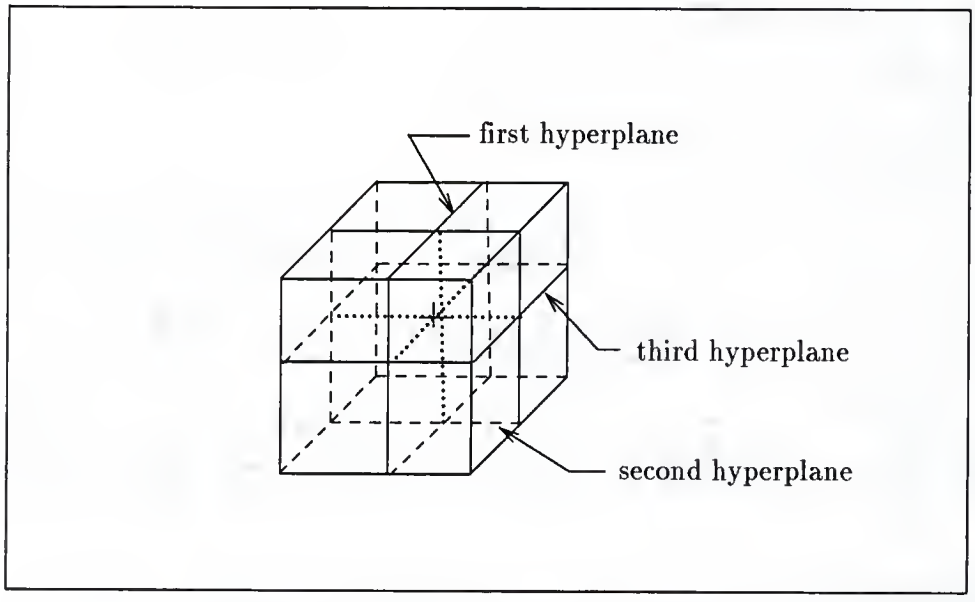

Figure 4. Three Solution Hyperplanes

shown in Figure 4. So solving a system of equations can be interpreted as finding a new set of hyperplanes that have the same intersection as the original set, but are also perpendicular to each other. If the problem that we are trying to solve is

the intersection of objects, then we have the additional constraint that we are only interested in solutions to the system of equations that lie inside the critical region. The critical region in this three-dimensional example is an unit cube with one corner at the origin and three of its edges lying along the three parameter axes. A solution to the system of equations is valid if, and only if, it lies inside the cube. Because the hyperplanes of the new system of equations are parallel to pairs of parameter axes, and therefore faces of the critical region cube, the solution lies inside the cube if, and only if, all of the hyperplanes in the new system pass through the critical region. This observation forms the basis of a criterion for intersection detection (assuming a constrained system of equations):

> two objects intersect if, and only if, the new system of hyperplanes formed such that all members are parallel to the facets of the critical region has the property that all hyperplanes pass through the critical region.

Underconstrained Systems

The criterion for intersection just stated will now be extended to underconstrained systems of equations. When a system of equations is underconstrained it is not possible to produce a new system that has each hyperplane parallel to a facet of the critical region. Instead we must settle for the hyperplanes being parallel to the edges of the facets. If there are p hyperplanes then we can find a new hyperplane which will be parallel to $p-1$ coordinate axes. That is each new hyperplane will be parallel to a $(p-1)$-face of the critical region. For example, if we have two equations in three unknowns then $n=3$, $p=2$, and

each new equation will have 1 zero coefficient and 2 non-zero coefficients. In terms of hyperplanes this means that the new hyperplanes will be parallel to the 1-faces of the cube rather than the facets.

If we attempt to solve an underconstrained system of equations in the same manner that we use for a constrained system we will find that it is impossible to form equations that have a single non-zero coefficient. Each new equation can be guaranteed to have at least $p-1$ zero coefficients, where the number of equations is p, out of a possible set of n coefficients. We must check every possible combination of $p-1$ zero coefficients out of the possible n coefficients. The number of new equations is $\binom{n}{p-1}$.

As an example, consider the intersection of two line segments of a one-dimensional space. The equations of intersection are

$$\begin{aligned}
&\lambda_1 P_1 + \lambda_2 P_2 - \lambda_3 P_3 - \lambda_4 P_4 = 0 \\
&\lambda_1 + \lambda_2 = 1 \\
&\lambda_3 + \lambda_4 = 1 \\
&\lambda_1, \lambda_2, \lambda_3, \lambda_4 \geq 0.
\end{aligned}$$

In this instance, we have three equations in four unknowns. Let a system of equation of this type be represented as

$$\begin{aligned}
a_1\lambda_1 + b_1\lambda_2 + c_1\lambda_3 + d_1\lambda_4 &= e_1 \\
a_2\lambda_1 + b_2\lambda_2 + c_2\lambda_3 + d_2\lambda_4 &= e_2 \\
a_3\lambda_1 + b_3\lambda_2 + c_3\lambda_3 + d_3\lambda_4 &= e_3
\end{aligned}$$

With three equations we can eliminate two coefficients from each of the new equations. Since there are four coefficients, the number of new equations is $\binom{4}{2} = 6$. The set of new equations is

$$
\begin{aligned}
0\lambda_1 &+ 0\lambda_2 &+ |cab|\lambda_3 &+ |dab|\lambda_4 &= |eab| \\
0\lambda_1 &+ |bac|\lambda_2 &+ 0\lambda_3 &+ |dac|\lambda_4 &= |eac| \\
0\lambda_1 &+ |bad|\lambda_2 &+ |cad|\lambda_3 &+ 0\lambda_4 &= |ead| \\
|abc|\lambda_1 &+ 0\lambda_2 &+ 0\lambda_3 &+ |dbc|\lambda_4 &= |ebc| \\
|abd|\lambda_1 &+ 0\lambda_2 &+ |cbd|\lambda_3 &+ 0\lambda_4 &= |ebd| \\
|acd|\lambda_1 &+ |bcd|\lambda_2 &+ 0\lambda_3 &+ 0\lambda_4 &= |ecd|
\end{aligned}
$$

The form of each coefficient follows a set pattern. The first letter is the column of the coefficient. The remaining letters are the columns of the zero terms for that equation. This might be called a generalization of Cramer's rule to underconstrained systems.

We now state the criterion of intersection for underconstrained systems.

> Given a set of $\binom{n}{p-1}$ equations, each of which represents a hyperplane parallel to $p-1$ axes, the common object of intersection passes through the critical region if, and only if, all of the hyperplanes represented by these equations pass through the critical region.

This needs to be proved. If the object of intersection passes through the critical region then each of the hyperplanes must also, since each must contain the object of intersection.

We need to prove that if the object of intersection does not pass through the critical region, then there must be at least one hyperplane which also does not pass through the critical region.

The intersection of any p hyperplanes represented by p independent equations from the set is the object of intersection in λ space. The dimension of

the object of intersection will be $n-p$. Assume that the object of intersection does not pass through the critical region. Each of the hyperplanes in the set must contain the object of intersection. Recall that each hyperplane in the set is parallel to $p-1$ axes. Since each hyperplane is parallel to $p-1$ axes, then there are $p-1$ coordinates which can change arbitrarily. Therefore a point in the plane can always be chosen such that $p-1$ of its coordinates are in the range zero to one. But the other $n-p+1$ coordinates must be such that the point lies in the object of intersection. Therefore, we must show that every point on the object of intersection is out of range for at least one of the $n-p+1$ coordinates if the hyperplane is to miss the critical region.

Let A be the object of intersection. A is an $(n-p)$-dimensional object in an n-dimensional space. Let A_i be the set of points in A with its i^{th} coordinate in range. Each of the A_i is a convex set. If we are to construct a hyperplane that does not pass through the critical region, then we need $n-p+1$ of these whose intersection is empty. We know that all n such spaces have an intersection that is empty since the object of intersection is assumed to miss the critical region. We have n convex sets in $(n-p)$-space. We want to show that if the intersection of all n is empty then there exists $n-p+1$ convex sets whose intersection is empty.

Helly's Theorem states that given a collection of convex sets in d-space, if any $d+1$ convex sets have an intersection that is non-empty, then all have an intersection that is non-empty [9, p.69]. Taking the reverse of Helly's theorem, if a collection of convex sets in d-space has an empty intersection, then there exists one choice of $d+1$ of the convex sets which must have an empty intersection.

The object of intersection is a space with dimension $d = n-p$. Since we know that all n convex sets, A_i, must have an empty intersection, then it must be true that $d+1 = n-p+1$ of the convex sets have an empty intersection. Since there is no point that has all of its coordinates in range the hyperplane must not pass through the critical region.

So, we need to check all of the hyperplanes which are parallel to the $n-p+1$ dimensional edges of the n-dimensional hypercube. Unfortunately there are $\binom{n}{p-1}$, that is $\frac{n!}{(p-1)!(n-p+1)!}$ of these hyperplanes. Each of these hyperplanes will have $p-1$ terms which are equal to zero and $n-p+1$ non-zero terms.

Checking for Intersection

The criterion developed depends on whether the hyperplanes pass through the critical region. A test for this condition will now be developed. Consider a three-dimensional parameter space. Here the critical region is a cube and the hyperplanes are planes. Assume that a given hyperplane is positioned so that it touches the critical region only in the point (1, 1, 1) as shown in Figure 5. The plane will be represented by an equation of the form

$$a\lambda_1 + b\lambda_2 + c\lambda_3 = d.$$

For our purposes we need to put the equation into a normalized form such that the coefficients a, b, c form a set of direction cosines and the sign of the right hand side is positive. The equation can now be written as

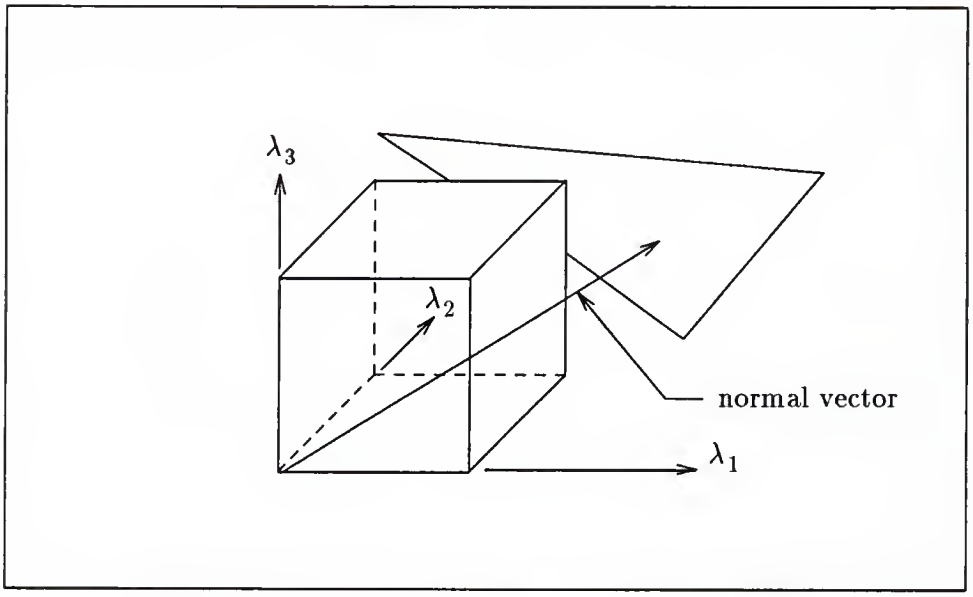

Figure 5. Plane Touching the Critical Region

$$\alpha\lambda_1 + \beta\lambda_2 + \gamma\lambda_3 = p.$$

In this form the coefficients of the λ's are the coefficients of the normal vector to the plane and p is the distance of the plane from the origin. If the plane touches the critical region at the point (1,1,1), this point must satisfy the equation. Substituting a value of one for each of the λ parameters yields

$$\alpha + \beta + \gamma = p.$$

This simple expression defines the condition that a plane just touch the critical region. If for a given plane the distance of the plane from the origin is less than the sum of the direction cosines of its normal vector then the plane intersects the critical region, else the plane does not intersect the critical region.

However, this statement assumes that the plane will touch the critical region first at the point (1,1,1). In terms of the direction cosines this is the condition that all of them are positive. The test can be made completely general

by choosing a new origin which is symmetrically located with respect to the corners of the critical region. The new origin will be at the point (0.5,0.5,0.5). The new coordinate system will then be scaled so that the cube has sides of length 2. In the new coordinate system all of the corners are located at a distance of $\sqrt{3}$ from the origin. The equation describing this transformation is

$$\lambda_i' = (\lambda_i - 0.5)2 = 2\lambda_i - 1$$

and the inverse transformation is

$$\lambda_i = \frac{(\lambda_i' + 1)}{2}.$$

Applying this transformation to the equation of a plane yields

$$a_1\lambda_1 + a_2\lambda_2 + a_3\lambda_3 = b$$

$$\frac{a_1(\lambda_1' + 1)}{2} + \frac{a_2(\lambda_2' + 1)}{2} + \frac{a_3(\lambda_3' + 1)}{2} = b$$

$$a_1(\lambda_1' + 1) + a_2(\lambda_2' + 1) + a_3(\lambda_3' + 1) = 2b$$

$$a_1\lambda_1' + a_2\lambda_2' + a_3\lambda_3' = 2b - (a_1 + a_2 + a_3).$$

This equation is then normalized so that the right hand side is positive. Since the critical region is symmetrical about the origin the equation can be converted to a first quadrant problem by taking the absolute value of each coefficient. Then the equation is evaluated to determine if the sum of the absolute values of the coefficients is less than the absolute value of the right hand side.

Embedded Problems and Overconstrained Problems

The method outlined above will solve the detection problem in general; however, there are certain special cases that require additional attention. These special cases can be divided into two groups: embedded problems and overconstrained problems.

Embedded problems are those problems for which the initial matrix of coefficients of the equations of intersection has a determinant of zero. This will occur when a lower-dimensional problem is embedded in a higher-dimensional space. For example, consider the intersection of two line segments in a plane as

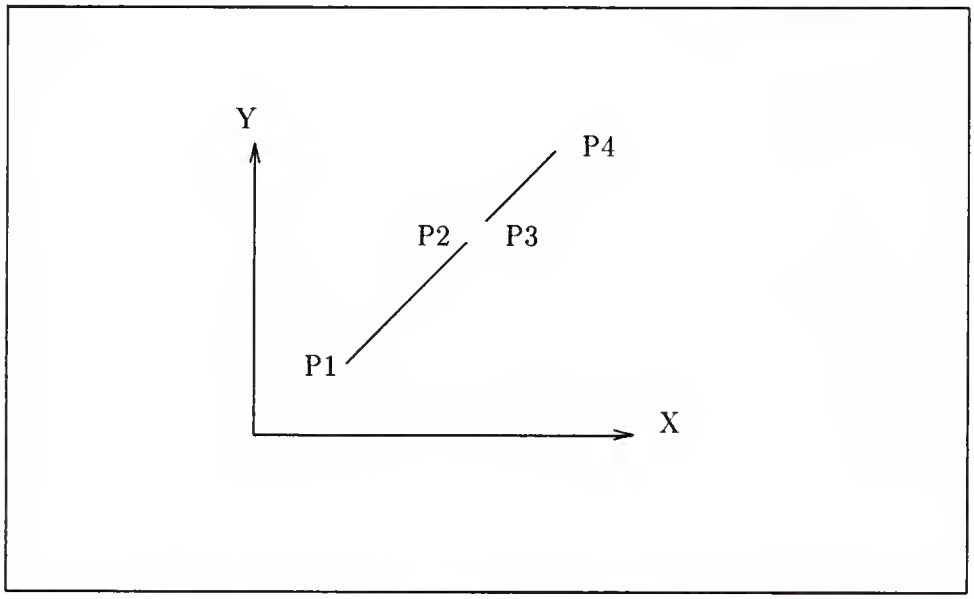

Figure 6. Two-Dimensional Embedded Problem

shown in Figure 6. If the two line segments are colinear, then this is really a one-dimensional problem, not a two-dimensional problem. This problem can be solved by reducing the dimension of the problem by examining the columns of the

matrix to find which are dependent and eliminating one column from the matrix of coefficients. Then the problem can be treated as a one-dimensional problem.

The second group of special cases are the overconstrained problems. An example of an overconstrained problem would be the intersection of two line segments in three space. There are four vertices but five equations, therefore the system is overconstrained. In three space two line segments can only intersect if they are coplanar. That is if there is an intersection then the problem must be an embedded problem. So, an overconstrained problem can only lead to an intersection if the rank of the matrix of coefficients and the rank of the augmented matrix of coefficients are equal and less than or equal to the number of unknowns. The rank of these two matrices can be determined by Gauss elimination. If the rank of the two matrices are equal to the number of unknowns then the single point of intersection can be determined by back substitution, and then checked to see if it lies within the critical region. If the rank of the two matrices are less than the number of unknowns then the problem is actually underconstrained and the reduced matrix produced by the Gauss elimination procedure can be used as a starting point for the method described above for intersection detection.

CHAPTER 7
DETERMINING INTERSECTIONS

Another Look at the New System

In detecting the occurrence of an intersection a new system of equations was produced. Each of the equations in this system has $p-1$ zero coefficients. However, in each equation the set of zero coefficients is different. The original set of p equations was linearly independent. Any p of the new equations will also be linearly independent, but any more than p must be linearly dependent. Choosing any p equations from the new set must define the object of intersection. But because the original system of equations was underconstrained any p equations will also be underconstrained. By choosing certain equations to form a set, and then setting the values of certain parameters it is possible to reduce the set to a solvable system of equations. The solution of a solvable system will be a vertex of the object of intersection. To find the object of intersection using this method it is required that we solve all sets of equations with a common set of $n-p$ parameters.

Solvable Equation Sub-Sets

The original system of p equations had n unknowns. From this system we produced a new system for which each equation has $p-1$ zero coefficients and in general the remaining $n-p+1$ coefficients are non-zero. We need to choose p of the equations to form a set. Recalling the solution of the constrained system of

equations, we would like to have equations which have only a single non-zero coefficient. But each of our equations has $n-p$ too many non-zero coefficients. Choose any set of $n-p$ parameters, say $\lambda_1, \lambda_2, \ldots, \lambda_{n-p}$. Select from the new system of equations all of the equations which have all of the $n-p$ parameters as non-zero coefficients. Each of these equations contains one other non-zero coefficient, and because they all contain $p-1$ different zero terms, each of these equations must contain one other different non-zero coefficient. This set of p equations in n unknowns gives us $n-p$ free choices. Let the free choices be to set the $n-p$ parameters which have the common non-zero coefficients to be equal to zero. Disregarding all terms which either have a zero coefficient or a zero parameter we now have p equations in p unknowns. Furthermore, each equation has only a single parameter with a non-zero coefficient. This set then leads directly to a point which is an element of the object of intersection.

As an example consider the one-dimensional case that we looked at in the last chapter. The new system of equations was

$$\begin{aligned}
0\lambda_1 + 0\lambda_2 + |cab|\lambda_3 + |dab|\lambda_4 &= |eab| \\
0\lambda_1 + |bac|\lambda_2 + 0\lambda_3 + |dac|\lambda_4 &= |eac| \\
0\lambda_1 + |bad|\lambda_2 + |cad|\lambda_3 + 0\lambda_4 &= |ead| \\
|abc|\lambda_1 + 0\lambda_2 + 0\lambda_3 + |dbc|\lambda_4 &= |ebc| \\
|abd|\lambda_1 + 0\lambda_2 + |cbd|\lambda_3 + 0\lambda_4 &= |ebd| \\
|acd|\lambda_1 + |bcd|\lambda_2 + 0\lambda_3 + 0\lambda_4 &= |ecd|
\end{aligned}$$

By examining each column of the set of equations it can be seen that each column has three non-zero terms, which is equal to the number of original equations. We select a set of three equations, each of which have the same non-zero coefficient, such as the last three equations in the set. These form a set of three equations in

four unknowns. So, we have one free choice to make in the solution of the system of equations. Let λ_1 be equal to zero. We can ignore the first column from the set of equations. What remains is now three equations in three unknowns, and furthermore, each equation contains only a single non-zero coefficient. To find the parameters of a point in the intersection all that remains to be done is to divide the right hand side of each of the three equations by the single coefficient. The values of the parameters would then be checked to see if they fall within the range zero to one. If they do then the point is a vertex of the object of intersection, if they do not then the point is discarded. This would be repeated for each the parameters.

The choice of setting all of the $n-p$ parameters to zero may seem arbitrary, but it does have a geometrical basis. Consider the mathematical definition of an object given in chapter four. Each object is defined as the convex combination of its extreme points. The value of each parameter is the contribution of its vertex to the position of a point. If any parameter is zero, then its vertex does not contribute, and the point must lie in the convex combination of the remaining vertices. As an example consider a triangle. If one of the parameters is zero, then the point must lie on the opposite side. So, choosing zero for the free parameter choices has the effect of causing the resulting point to lie on the edges of the objects.

Considering the geometric interpretation of the terms allows further insight into the solution. The process will always result in a number of non-zero terms that depends only on the dimension of the problem. In a one-dimensional

problem the number of non-zero terms is always three. The number of parameters can at most be equal to four, because each one corresponds to one of the four points which define the two segments of the one-dimensional space. Therefore, taking three points, one of the points must be from one segment and the other two points from the other segment. Since the sum of the parameter values for any object must be equal to one, the value of the single parameter must be equal to one, and the sum of the two parameters must be equal to one. In addition, the parameters from the two different objects must indicate the same point. So the solution must indicate the parameters of the point that corresponds to the single parameter in terms of the basis defined by the pair of points from the second object. In a two-dimensional problem the number of non-zero terms is always four. If one parameter is from the first object and three from the second object, then the solution indicates the parameters of the point from the first object in terms of the basis of three points from the second object. If two parameters are from the first object and two from the second object, then the solution indicates the parameters of the point of intersection of the two line segments that connect the pairs of points from each object. In a three-dimensional problem the number of non-zero terms is always five. If one parameter is from the first object and four from the second object, then the solution indicates the parameters of the point from the first object in terms of the basis of four points from the second object. If two parameters are from the first object and three from the second object, then the solution indicates the parameters of the point where the line through the two points intersects the plane

through the three points in terms of the basis of three points and in terms of the basis of two points.

Elimination of Non-Extreme Points

If we evaluate every set of equations that have a common set of $n-p$ non-zero parameters we are guaranteed to find all of the extreme points of the object of intersection. This process will also find other points which are members of the object of intersection but are not extreme points if one or both of the objects

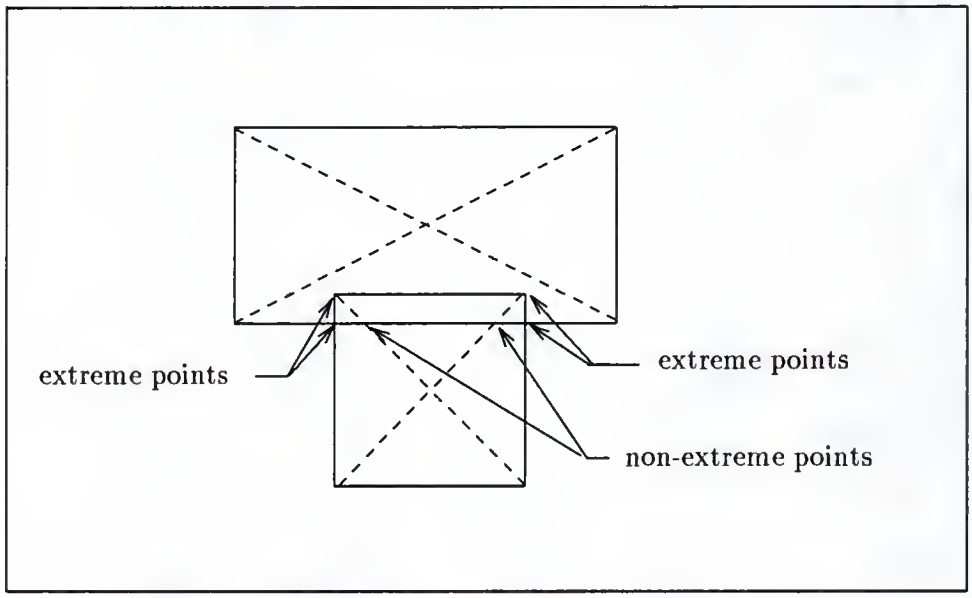

Figure 7. Intersection Containing Non-Extreme Points

contain more than $d+1$ points. Consider a square as shown in Figure 7. If the edges are considered to be every combination of two points then there are six edges. Four edges form the boundary of the square and two edges form the diagonals. When the square is intersected with another object these diagonal edges will give intersection points that are valid elements of the intersection set

but which are not extreme points. The object facial structure list is used to produce only those sets of $n-p$ parameters that will yield extreme points. In a one-dimensional problem each set of three is formed by taking one parameter from one object and two from the other. In a two-dimensional problem each set of four parameters is formed by pairing each edge of one object with every edge from the other object, and then each point of one object with each face of the other object. In a three-dimensional problem each set of five parameters is formed by taking each edge of each object with each face of the other object, and then each point of each object with every volume of the other object.

Object Facial Structure List Construction

Since an object now consists of a list of vertices and an object facial structure list, if our algorithm is to produce an object definition as its output then we must be able to construct an object facial structure list for the new object of intersection.

In one dimension the list is trivial since there can only be a single edge.

In two dimensions the list can be constructed if the points are listed in a cyclic order. This ordering of the points can be achieved by finding the centroid of the vertices and the vectors from the centroid to each vertex. Then choosing one vector as the reference vector, that is the vector that points in direction zero, the relative directions of all of the other vectors can be found by taking the scalar and vector products of each of them with the reference vector. Having assigned to each point an angle they may be ordered in a cyclic list. The edges can then be listed by taking each pair of points in succession. The object can be divided

up into non-overlapping triangles by taking the first point in the cyclic list as a pivot along with each set of two of the other points in succession, that is 123, 134, 145, and so on.

In three dimensions finding the edges and faces requires that we determine which of the new points lie on each of the composite faces of each of the old objects. Once the set of points on a composite face are known they can be treated in a manner similar to that described for the two-dimensional case resulting in a list of edges and a list of faces. For each composite face all of the new faces found become a new composite face. Taking each composite face in turn a complete list of edges, faces, and composite faces can be built for the new object. The three-dimensional object must also be divided into non-overlapping volumes. This can be accomplished as follows. Find two points which are not on any common composite face. Find the normal vector of each plane formed by these two points and each of the other points in the object. If any of the normal vectors point in the same direction choose a different pair of starting points. (This method may fail in certain cases because it may not be possible to find a suitable pair of points for particular objects. A general method of dividing an object into tetrahedrons is needed but that is beyond the scope of this work.) Each of these normal vectors must be perpendicular to the line through the two points, and being free vectors may be considered to be coplanar. Choosing the first vector as a reference pointing in direction zero the vectors may be ordered in a cyclic manner by calculating the relative angels between each of them and the reference. Since the normal vectors can be ordered then the points which determine the planes can be

ordered. Having ordered the set of points in the object it can now be divided into tetrahedrons by taking the two points along with each pair of points from the cyclic list.

Transformation to Coordinate Space

The final step is to transform the points from the parameter space back into the coordinate space. The result of the solution of the sets of equations is in terms of the λ-parameters. We want the points to be given in terms of the xyz-coordinates. This is accomplished by multiplying the coordinates of one object by the parameters for those points produced by the solution method. The solution method gives the solution in terms of the parameters of both objects. Each set of parameters giving a single intersection point indicates the solution in terms of both objects. If the number of vertices in the first object is v1, then the first v1 parameters give the position of the point of intersection as a convex combination of these v1 vertices. Likewise, if the second object has v2 vertices, then the last v2 parameters give the position of the point of intersection as a convex combination of the vertices of the second object. To find the coordinates of a point of intersection choose one object and select the parameters for that object from the list of parameters. Then multiply each of the vertices by the corresponding parameter and then take the summation of each coordinate. The resulting vector is the desired point.

CHAPTER 8
IMPLEMENTATION

This chapter discusses an implementation of the algorithms presented. The results show the relative cost of the algorithm and show where the most improvement is needed to reduce the cost. This is not a definitive evaluation since the results depend upon the hardware and the particular rendering of the algorithms into computer code. The programs were written to show that the algorithms do work but they were not optimized for speed or storage requirements and therefore do not represent the lowest possible cost for the algorithm.

Programming Environment

The computer used for the program development was a 33-Mhz 80386 PC compatible with a 80387 co-processor. The operating system was Interactive Unix version 2.2. The programming language used was the C language and was compiled by the standard Unix compiler.

The Programs

A set of functions has been written, each of which performs a step in the solution. This section is a brief outline of the functions and the sequence in which they are called.

The function form_system() takes two object data structures and creates the system of equations of intersection. The data concerning a single object was

stored in a C data structure as shown in Figure 8. The function assumes that the data structures are filled with data that properly define convex objects. No checking is done to see if the data are correct and the results are undefined if the

```
typedef struct object {
                int     d;
                int     v;
                double  point[20][3];
                int     edges;
                int     edge[20][2];
                int     faces;
                int     face[20][3];
                int     volms;
                int     volm[10][4];
                int     comfs;
                int     comf[10][12];
              } Object;
```

Figure 8. The Definition of the C Data Structure for an Object

data are incorrect. The resulting system of equations is stored in an array and passed back to the calling program.

The function new_sys() takes the system of equations of intersection produced by the form_sys() function and from it forms the new system of equations each of which contain the maximum number of zero coefficients. For each new equation the coefficients are calculated and the equation is checked to see if the hyperplane that it represents passes through the critical region. If any of the hyperplanes represented by one of the new equations does not pass through the critical region then new_sys() returns a value of zero, indicating that the two objects do not intersect. Otherwise it returns a value of one, indicating that the two objects do intersect, and passes the coefficients of the new system of

equations back to the calling program in an array. This completes the detection part of the problem.

If the two objects do intersect then a new object is created containing the object of intersection. The function fnd_vrt() is called to find the values of the parameters of the vertices of the object of intersection. The function unq_vrt() is called to eliminate duplicate vertices. The function frm_obj() is called to transform the parameters of the vertices into coordinates in xyz-space. Finally the function bld_lst() is called to create the facial structure list for the new object. The result is an object data structure filled with the data for the object of intersection. This completes the determination part of the problem.

Computational Cost

The Form of the Test

In order to judge the relative computational cost of the implementation of the algorithms a set of time trials was conducted. A set of pairs of objects was constructed such that each pair contained an increasing total number of vertices. In all of the pairs of objects there was an intersection so that all of the functions would be called. A program was written that called each of the functions named above in the proper order. The Unix system call times() was used to determine the cumulative cpu time used by the program. By calling the times() function before and after each of the function calls and then taking the difference, the amount of cpu time used by each function was found. Since the resolution of the times() function was one-hundredth of a second, each of the functions was placed

within a for loop and called 100 times. The difference between the cumulative cpu time before and after the loop was then divided by 100 to get an average time needed for one execution of the function. Time trials were run separately for objects of two- and three-dimensions.

Two-Dimensional Results

Table 1 shows the results of the two-dimensional trials. As the table shows the two functions with the highest cost are new_sys() that forms the new system

Table 1. Average CPU Time

	cpu time in milliseconds						
number of old vertices	form_sys	new_sys	fnd_vrt	unique_vrt	form_obj	build_list	number of new vertices
4	0.20	3.10	0.20	0.00	0.40	0.40	1
5	0.30	10.30	2.80	0.10	0.30	0.20	2
6	0.30	26.10	14.80	0.20	0.40	1.30	4
7	0.30	55.70	43.90	0.40	0.60	1.90	6
8	0.40	102.30	118.10	0.60	0.80	2.70	8
9	0.30	177.30	273.30	1.10	1.00	3.00	9
10	0.30	283.50	705.00	1.10	1.30	3.40	10

of equations and detects whether an intersection occurs and fnd_vrt() that calculates the values of the parameters of the vertices of the new object of intersection. Figure 9 shows a graph of the cost of the function new_sys() as a function of the total number of vertices in the two original objects. Figure 10 shows a graph of the cost of the function fnd_vrt() as a function of the total number of vertices in the two original objects. These two plots illustrate the rate at which the cost of the two functions increase as the size of the problem increases for a two-dimensional problem. As the table and plots show the function with

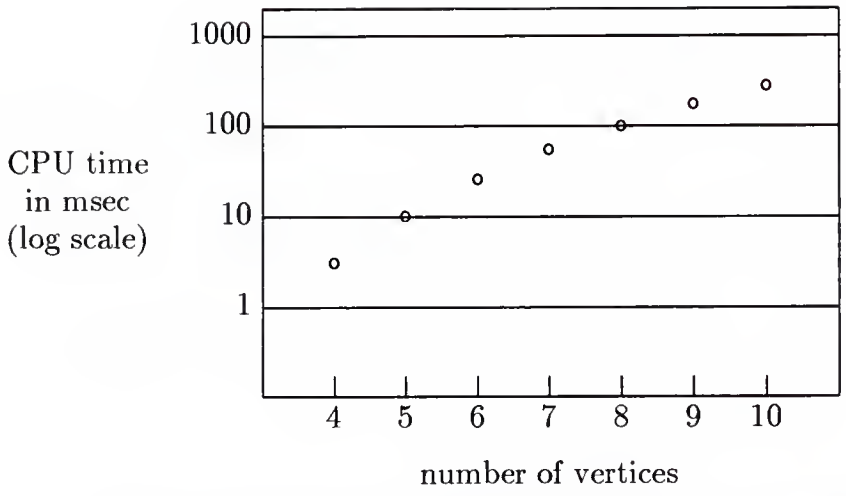

Figure 9. Cost of Forming the New System of Equations as a Function of Number of Vertices in Two-Dimensions

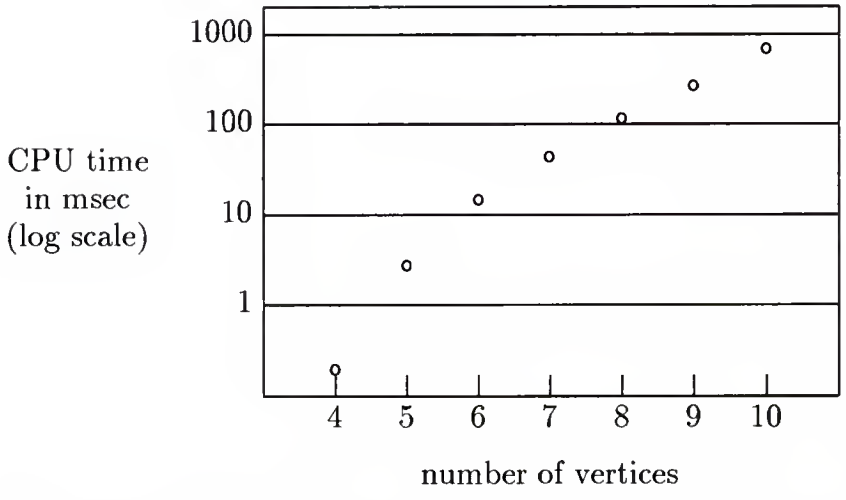

Figure 10. Cost of Finding the New Vertices of the Object of Intersection as a Function of Number of Vertices in Two-Dimensions

the greatest cost is the fnd_vrt() function which finds the parameters for each of the vertices of the new object of intersection.

Three-dimensional Results

Table 2 shows the results of the three-dimensional trials. As in the two-dimensional case the two functions that have the greatest cost are new_sys() and fnd_vrt(). Figure 11 shows a graph of the cost of the function new_sys() as a

Table 2. Average CPU Time

	cpu time in milliseconds						
number of old vertices	form_sys	new_sys	fnd_vrt	unique_vrt	form_obj	build_list	number of new vertices
5	0.30	15.60	0.20	0.00	0.30	0.20	1
6	0.30	67.30	4.80	0.10	0.50	0.10	2
7	0.30	208.80	30.70	0.30	0.50	0.10	3
8	0.40	502.70	184.20	0.20	0.80	0.20	4
9	0.40	1073.70	563.20	1.20	1.30	10.10	7

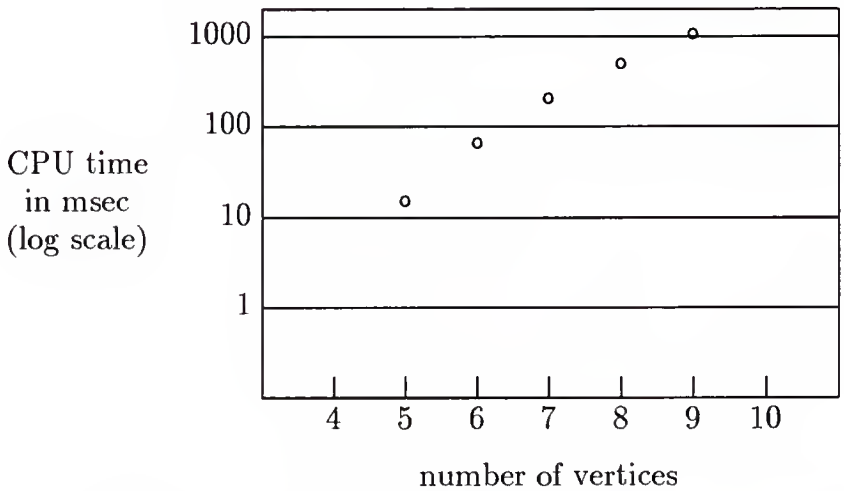

Figure 11. Cost of Forming the New System of Equations as a Function of Number of Vertices in Three-Dimensions

function of the total number of vertices in the two original objects. Figure 12 shows a graph of the cost of the function fnd_vrt() as a function of the total number of vertices in the two original objects. These two plots illustrate the rate at which the cost of the two functions increase as the size of the problem increases for a three-dimensional problem. As the table and graphs show the function with the greatest cost is the new_sys() function which forms the new system of equations.

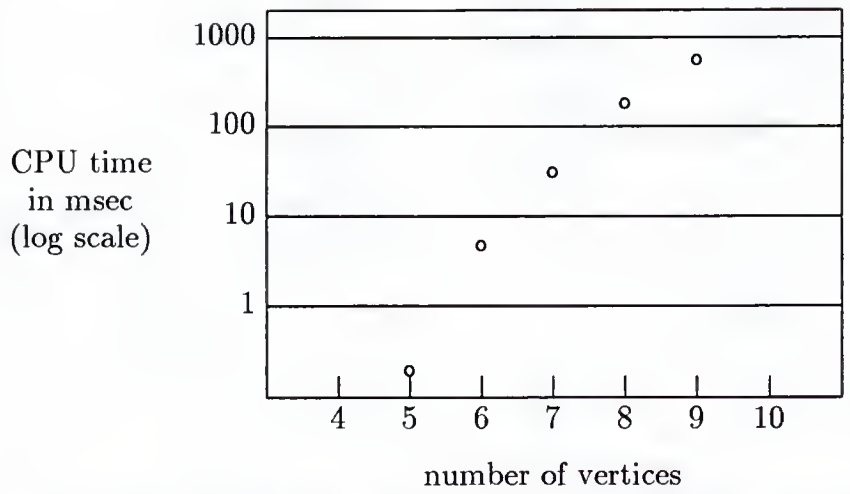

Figure 12. Cost of Finding the New Vertices of the Object of Intersection as a Function of Number of Vertices in Three-Dimensions

These results show that the two functions new_sys() and fnd_vrt() are the ones that need the most improvement. The current implementation is not fast enough for a real time application and is probably too slow for an interactive application. However, the current implementation does not represent the lowest cost that the application can achieve. By optimizing the computer programs used to implement the algorithm lower cost determination and detection can be achieved.

CHAPTER 9
CONCLUDING REMARKS

Summary

A method for detecting the intersection of convex objects in two- and three-dimensional space which have a finite number of extreme points has been presented. The method was based on a definition of an object that was drawn from the theory of convex sets. Using this definition, the conditions of intersection were expressed as a set of linear equations and a set of inequalities. From this set of equations a new set of equations was produced which allowed for the existence of an intersection to be detected. The method was extended to determine the object of intersection. The result of the determination is a list of the extreme points and lists describing the facial structure of the object of intersection.

Suggestions for Further Work

There are two areas that are open for further investigation: improving the implementation to achieve faster detection and determination, and investigating the nature of the new system of equations to gain further insight into the intersection problem.

A better implementation of the algorithm should be able to reduce the computational cost of the algorithm. One place where the implementation can be

improved is in the production of the new system of equations. In the current implementation each term is calculated by evaluating a determinant, as would be done if a system of equations was solved by Cramer's rule. A new implementation which uses a more efficient method to produce the new system of equations would reduce the cost of the algorithm.

The new system of equations must contain information about the relative position and orientation of the two objects. It may be possible to extract this information in forms that are pertinent to various applications. It has been shown that intersection is detected by the intersection of the hyperplanes that are represented by the members of the new system of equations with the critical region. It is possible to determine the distance between each hyperplane and the nearest corner of the critical region. For instance, it may be possible to produce a metric that indicates the "distance" between two objects. Or, it may be possible to determine the orientation that allows an object to pass between two obstacles. The relationship between the relative positions and orientations of two objects and the associated new system of equations should be investigated.

REFERENCES

1. Boyse, J. W., "Interference Detection Among Solids and Surfaces," *Communications of the ACM,* Vol. 22, No. 1, 1979, pp. 3-9.

2. Chazell, B. and Dobkin, D. P., "Intersection of Convex Objects in Two and Three Dimensions," *Journal of the Association for Computing Machinery,* Vol. 34, No. 1, 1987, pp. 1-27.

3. Shamos, M. I., and Hoey, D., "Geometric Intersection Problems," *Proc. of the 17 IEEE Symp. on Foundations of Computer Science,* 1976, pp. 208-215.

4. Nievergelt, J. and Preparata, F. P., "Plane-Sweep Algorithms for Intersecting Geometric Figures," *Communications of the ACM,* Vol. 25, No. 10, 1982, pp. 739-747.

5. Muller, D. E., and Preparata, F. P., "Finding the Intersection of Two Convex Polyhedra," *Theoretical Computer Science,* Vol. 7, 1987, pp. 217-236.

6. Dobkin, D. P. and Kirkpatrick, D. G., "Fast Detection of Polyhedral Intersection," *Theoretical Computer Science,* Vol. 27, 1983, pp. 241-253.

7. Ganter, M. A. and Uicker, J. J. Jr., "Dynamic Collision Detection Using Swept Solids," *Journal of Mechanisms, Transmissions, and Automation in Design,* Vol. 108, 1986, pp. 549-555.

8. Lozano-Perez, T., "Automatic Planning of Manipulator Transfer Movements," *IEEE Transactions on System, Man, and Cybernetics,* Vol. SMC-11, No. 10, 1981, pp. 681-689.

9. Brondsted, A., *An Introduction to Convex Polytopes,* Springer-Verlag, New York, 1982.

10. Valentine, F. A., *Convex Sets,* McGraw-Hill, New York, 1964.

BIOGRAPHICAL SKETCH

Paul Jeffrey Lewicki was born in Rochester, Pennsylvania, on March 22, 1961. He moved to Panama City, Florida, in 1972. He began undergraduate studies at the University of Florida in January, 1980. He earned a Bachelor of Science in Mechanical Engineering degree in May, 1982 and a Master of Engineering degree in May, 1986, both from the University of Florida.

I certify that I have read this study and that in my opinion it conforms to acceptable standards of scholarly presentation and is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.

Gary K. Matthew, Chairman
Associate Professor of
Mechanical Engineering

I certify that I have read this study and that in my opinion it conforms to acceptable standards of scholarly presentation and is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.

Joseph Duffy
Graduate Research Professor of
Mechanical Engineering

I certify that I have read this study and that in my opinion it conforms to acceptable standards of scholarly presentation and is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.

Carl D. Crane
Assistant Professor of
Mechanical Engineering

I certify that I have read this study and that in my opinion it conforms to acceptable standards of scholarly presentation and is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.

Ralph G. Selfridge
Professor of
Computer and Information Sciences

I certify that I have read this study and that in my opinion it conforms to acceptable standards of scholarly presentation and is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.

Neil L. White
Professor of
Mathematics

This dissertation was submitted to the Graduate Faculty of the College of Engineering and to the Graduate School and was accepted as partial fulfillment of the requirements for the degree of Doctor of Philosophy.

December, 1991

for Winfred M. Phillips
Dean, College of Engineering

Madelyn M. Lockhart
Dean, Graduate School

UNIVERSITY OF FLORIDA
3 1262 08285 458 8